Une nouvelle métaheuristique pour l'optimisation difficile : la méthode des essaims particulaires

Maurice Clerc* et Patrick Siarry**

* France Télécom R&D
12 av. de Chevêne, 74988 Annecy
maurice.clerc@writeme.com

** Université de Paris 12 Val-de-Marne, LERISS (E.A. 412)
61 av. du Général de Gaulle, 94010 Créteil
siarry@univ-paris12.fr

Résumé

Nous présentons d'abord dans cet article le cadre général des métaheuristiques, qui sont apparues au début des années 1980 avec une ambition commune : résoudre au mieux les problèmes dits d'optimisation difficile. Nous essayons de cerner le domaine de l'optimisation difficile, tant pour les problèmes discrets que pour les problèmes à variables continues. Puis nous dégageons quelques caractéristiques communes à nombre de métaheuristiques, et nous passons en revue les principales extensions de ces méthodes.
Nous présentons ensuite une métaheuristique apparue dernièrement : la méthode d'optimisation par essaim particulaire (OEP). L'OEP est une technique encore peu connue en France, fondée sur la notion de coopération entre des agents (les particules) qui peuvent être vus comme des « animaux » aux capacités assez limitées (peu de mémoire et de facultés de raisonnement). L'échange d'information entre eux fait que, globalement, ils arrivent néanmoins à résoudre des problèmes difficiles, comme c'est le cas, par exemple, chez les abeilles vivant en essaim (exploitation de sources de nourriture, construction de rayons, etc.). Après une présentation succincte des origines, l'article propose une description informelle de l'OEP, puis en dégage les principales caractéristiques. Simple à comprendre, à programmer et à utiliser, l'OEP se révèle particulièrement efficace pour les problèmes d'optimisation non linéaire, à variables continues, entières ou mixtes.
L'article se termine par un résumé des exposés présentés lors du premier séminaire francophone sur le sujet (OEP'2003. Paris, octobre 2003).

Mots clés : optimisation, métaheuristiques, essaims particulaires.

1. Les métaheuristiques pour l'optimisation difficile

1.1 Optimisation « difficile »

Les ingénieurs et les décideurs sont confrontés quotidiennement à des problèmes de complexité grandissante, qui surgissent dans des secteurs techniques très divers, comme dans la conception de systèmes mécaniques, le traitement des images, l'électronique ou la recherche opérationnelle. Le problème à résoudre peut souvent s'exprimer comme un *problème d'optimisation* : on définit une fonction objectif, ou fonction de coût (voire plusieurs), que l'on cherche à minimiser ou à maximiser par rapport à tous les paramètres

concernés. La définition du problème d'optimisation est souvent complétée par la donnée de *contraintes*. On distingue les contraintes *impératives* (ou « dures ») et les contraintes *indicatives* (ou « molles »). Tous les paramètres des solutions retenues doivent respecter les contraintes du premier type, qui peuvent aussi être vues comme définissant l'espace de recherche, faute de quoi ces solutions ne sont pas réalisables. Les contraintes indicatives doivent simplement être respectées aussi bien que possible. De nouvelles méthodes, dénommées *métaheuristiques*, comprenant notamment la méthode du recuit simulé, les algorithmes évolutionnaires, la méthode de recherche tabou, les algorithmes de colonies de fourmis... sont apparues, à partir des années 1980, avec une ambition commune : résoudre *au mieux* les problèmes dits d'*optimisation difficile*.

Pour tenter de cerner le domaine – mal défini – de l'optimisation difficile, il est nécessaire de faire la distinction entre deux types de problèmes d'optimisation : les problèmes « discrets » et les problèmes à variables continues. Citons un exemple de chaque type, pour fixer les idées. Parmi les problèmes discrets, on trouve le célèbre problème du voyageur de commerce : il s'agit de minimiser la longueur de la tournée d'un « voyageur de commerce », qui doit visiter un certain nombre de villes, avant de retourner à la ville de départ. Dans la catégorie des problèmes continus, un exemple classique est celui de la recherche des valeurs à affecter aux paramètres d'un modèle numérique de processus, pour que ce modèle reproduise au mieux le comportement réel observé. En pratique, on rencontre aussi des « problèmes mixtes », qui comportent à la fois des variables discrètes et des variables continues.

Revenons sur la définition de l'optimisation difficile. Deux sortes de problèmes reçoivent, dans la littérature, cette appellation, non définie strictement (et liée, en fait, à l'état de l'art en matière d'optimisation) :

- certains problèmes d'optimisation discrète, pour lesquels on ne connaît pas d'algorithme exact *polynomial* (c'est-à-dire dont le temps de calcul est proportionnel à N^n, où N désigne le nombre de paramètres inconnus du problème, et n est une constante entière). C'est le cas, en particulier, des problèmes dits « NP-difficiles », pour lesquels on conjecture qu'il n'existe pas de constante n telle que le temps de résolution soit borné par un polynôme de degré n.
- certains problèmes d'optimisation à variables continues, pour lesquels on ne connaît pas d'algorithme permettant de repérer un *optimum global* (c'est-à-dire la meilleure solution possible) à coup sûr et en un nombre fini de calculs.

Des efforts ont longtemps été menés, séparément, pour résoudre ces deux types de problèmes. Dans le domaine de l'optimisation continue, il existe ainsi un arsenal important de méthodes classiques dites d'*optimisation globale* [Berthiau et al. 01], mais ces techniques sont souvent inefficaces si la fonction objectif ne possède pas une propriété structurelle particulière, telle que la convexité. Dans le domaine de l'optimisation discrète, un grand nombre d'*heuristiques*, qui produisent des solutions proches de l'optimum, ont été développées ; mais la plupart d'entre elles ont été conçues spécifiquement pour un problème donné.

1.2 Cadre des métaheuristiques

L'arrivée des *métaheuristiques* marque une réconciliation des deux domaines : en effet, celles-ci s'appliquent à toutes sortes de problèmes discrets, et elles peuvent s'adapter aussi aux problèmes continus. Ces méthodes ont en commun, en outre, les caractéristiques suivantes :

- elles sont, au moins pour partie, *stochastiques* : cette approche permet de faire face à l'*explosion combinatoire* des possibilités ;
- souvent d'origine discrète (à l'exception notable de l'OEP), elles ont l'avantage, décisif dans le cas continu, d'être *directes*, c'est-à-dire qu'elles ne recourent pas au calcul, souvent problématique, des gradients de la fonction objectif ;
- elles sont inspirées par des *analogies* : avec la physique (recuit simulé, diffusion simulée...), avec la biologie (algorithmes évolutionnaires, recherche tabou...) ou avec l'éthologie (colonies de fourmis...) ;
- elles partagent aussi les mêmes inconvénients : les difficultés de *réglage* des paramètres de la méthode, et le *temps de calcul* élevé.

Ces méthodes ne s'excluent pas mutuellement : en effet, dans l'état actuel de la recherche, il est le plus souvent impossible de prévoir avec certitude l'efficacité d'une méthode donnée, quand elle est appliquée à un problème donné. Cette propriété déconcertante a même fait l'objet d'une tentative de formalisation avec les théorèmes dénommés « No free lunch » [Wolpert et al. 97]. Un sujet théorique de recherche concerne les possibilités d'analyse systématique des métaheuristiques du point de vue de la convergence, la complexité, la robustesse et les garanties de qualité ; une voie prometteuse est l'analyse des *paysages d'énergie* (rugosité, caractère fractal…).

De plus, la tendance actuelle est l'émergence de *méthodes hybrides*, qui s'efforcent de tirer parti des avantages spécifiques d'approches différentes en les combinant. E.G. Talbi propose ainsi une taxinomie des métaheuristiques hybrides [Talbi 02]. La coopération entre métaheuristiques peut aussi prendre la forme de systèmes multi-agents, capables en principe de changer automatiquement d'outil de recherche, en fonction des difficultés rencontrées.

Enfin, dans leur démarche unificatrice de la *programmation à mémoire adaptative* [Taillard et al. 98], E. Taillard et al. remarquent que les métaheuristiques reposent finalement sur un ensemble commun de concepts peu nombreux : la *mémoire*, qui sauvegarde l'information recueillie par l'algorithme, l'*intensification*, qui tente d'améliorer la pertinence des informations disponibles, au moyen de recherches locales, et la *diversification*, qui vise à accroître la quantité de ces informations, en explorant de nouvelles régions de l'espace de recherche. Les associations multiples de ces concepts peuvent conduire à une grande variété de métaheuristiques : la filiation à une métaphore donnée n'est alors plus nécessairement explicite.

1.3 Extensions

Pour compléter cette présentation succincte du domaine, on peut souligner une autre richesse des métaheuristiques : elles se prêtent à toutes sortes d'*extensions*. Citons, en particulier :

- l'optimisation *multiobjectif* [Collette et al. 02], où il s'agit d'optimiser simultanément plusieurs objectifs contradictoires. Il n'existe pas, dans ce cas, un optimum unique ; on cherche, en revanche, une gamme de solutions « optimales au sens de Pareto », qui forment la « surface de compromis » du problème considéré. Ces solutions peuvent être soumises à l'arbitrage final de l'utilisateur ;
- l'optimisation *multimodale*, où l'on s'efforce de repérer tout un jeu d'optimums globaux ou locaux. Les algorithmes évolutionnaires sont particulièrement bien adaptés à cette tâche, de par leur nature distribuée. Les variantes de type « multipopulation »

exploitent en parallèle plusieurs populations, qui s'attachent à repérer des optimums différents ;

- le recours à des *implémentations parallèles*. De multiples modes de parallélisation ont été proposés pour les différentes métaheuristiques. Certaines techniques se veulent générales ; d'autres, en revanche, tirent parti de particularités du problème. Ainsi, dans les problèmes de placement de composants, les tâches peuvent être réparties naturellement entre plusieurs processeurs : chacun d'eux est chargé d'optimiser une zone géographique donnée, et des informations sont échangées périodiquement entre processeurs voisins ;
- l'optimisation *dynamique*, qui fait face à des variations temporelles de la fonction objectif au cours de l'optimisation : il faut alors approcher au mieux la solution optimale à chaque pas de temps. Les problèmes d'optimisation dynamique semblent être la cible de choix pour les algorithmes de colonies de fourmis et d'essaims particulaires, en particulier quand on ne dispose que d'informations locales. La structure distribuée et le caractère auto-organisé de ces algorithmes leur procurent en effet la flexibilité nécessaire pour évoluer dans un environnement changeant.

On voit que ces contextes particuliers requièrent, de la part des méthodes de résolution, des propriétés spécifiques qui ne sont pas présentes dans toutes les métaheuristiques ; de là peut découler un moyen de guidage de l'utilisateur dans le choix d'une métaheuristique.

1.4 Un sujet ouvert : le choix d'une métaheuristique

Cette présentation ne doit pas éluder la principale difficulté à laquelle est confronté l'ingénieur, en présence d'un problème d'optimisation concret : celui du choix d'une méthode « efficace », capable de produire une solution « optimale » - ou de qualité acceptable – au prix d'un temps de calcul « raisonnable ». Face à ce souci pragmatique de l'utilisateur, la théorie n'est pas encore d'un grand secours, car les théorèmes de convergence sont souvent inexistants ou applicables seulement sous des hypothèses très restrictives. En outre, le réglage « optimal » des divers paramètres d'une métaheuristique, qui peut être préconisé par la théorie, est souvent inapplicable en pratique, car il induit un coût de calcul prohibitif. En conséquence, le choix d'une « bonne » méthode, et le réglage des paramètres de celle-ci, font généralement appel au savoir-faire et à l' « expérience » de l'utilisateur, plutôt qu'à l'application fidèle de règles bien établies. Les efforts de recherche en cours, par exemple sur l'analyse des « paysages d'énergie » ou sur la mise au point d'une taxinomie des méthodes hybrides, visent à remédier à cette situation, périlleuse à terme pour la crédibilité des métaheuristiques….

Le réglage et la comparaison des métaheuristiques sont souvent effectués empiriquement, en exploitant des jeux de fonctions analytiques de test, dont les minimums globaux et locaux sont connus. Nous donnons, à titre d'exemple, en figure 1, l'allure de l'une de ces fonctions de test.

1.5 Pour en savoir plus sur les métaheuristiques

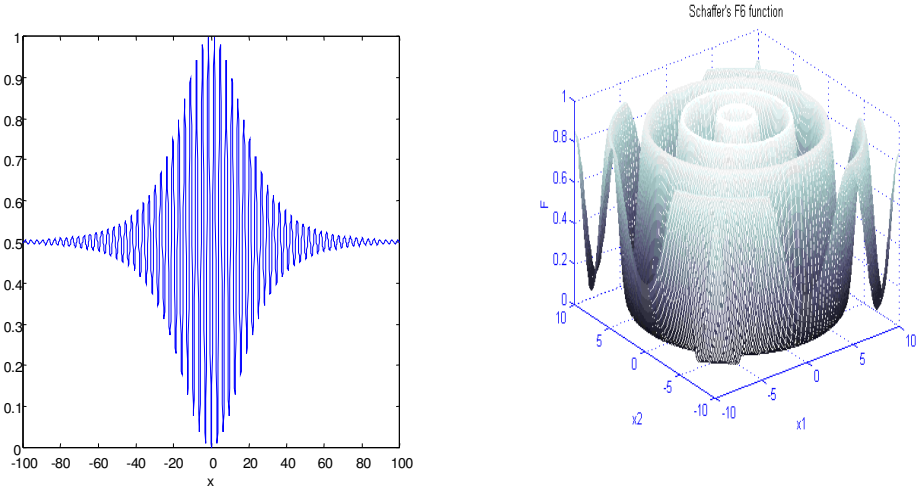
Le lecteur intéressé pourra consulter un ouvrage de synthèse récent : [Dréo et al. 03], qui a été coordonné par l'un des auteurs de cet article, ainsi que les ouvrages : [Reeves 95] [Saït et al. 99] [Pham et al. 00] [Pirlot et al. 02] et [Pirlot et al. 03] décrivant plusieurs métaheuristiques. Il existe aussi des livres en français consacrés à une seule méthode, comme [Siarry et al. 89]

pour le recuit simulé et [Goldberg 94] pour les algorithmes génétiques, et un livre de référence (en anglais) sur la recherche tabou [Glover et al. 97].

2. L'optimisation par essaim particulaire

2.1 Origines

L'optimisation par essaim particulaire (OEP) est une méthode née en 1995 aux Etats-Unis sous le nom *de Particle Swarm Optimization (PSO)* .



(a) (b)

(a) représentation à une dimension dans le domaine [-100, 100]
(b) représentation à deux dimensions dans le domaine [-10, 10]

Figure 1. *Allure de la fonction de test F6.*

Initialement, ses deux concepteurs, Russel Eberhart et James Kennedy, cherchaient à modéliser des interactions sociales entre des « agents » devant atteindre un objectif donné dans un espace de recherche commun, chaque agent ayant une certaine capacité de mémorisation et de traitement de l'information. La règle de base était qu'il ne devait y avoir aucun chef d'orchestre, ni même aucune connaissance par les agents de l'ensemble des informations, seulement des connaissances locales. Un modèle simple fut alors élaboré.
Dès les premières simulations, le comportement collectif de ces agents évoquait celui d'un essaim d'êtres vivants convergeant parfois en plusieurs sous-essaims vers des sites intéressants. Ce comportement se retrouve dans bien d'autres modèles, explicitement inspirés des systèmes naturels (cf. par exemple). Ici, la métaphore la plus pertinente est probablement celle de l'essaim d'abeilles, particulièrement du fait qu'une abeille ayant trouvé un site prometteur sait en informer certaines de ses consœurs et que celles-ci vont tenir compte de cette information pour leur prochain déplacement . Finalement, le modèle s'est révélé être trop simple pour vraiment simuler un comportement social, mais par contre très efficace en tant qu'outil d'optimisation.

Comme nous allons le voir, le fonctionnement de l'OEP fait qu'elle peut être rangée dans les méthodes itératives (on approche peu à peu de la solution) et stochastiques (on fait appel au

hasard). Sous ce terme un peu technique, on retrouve un comportement qui est aussi vieux que la vie elle-même : améliorer sa situation en se déplaçant partiellement au hasard et partiellement selon des règles prédéfinies.

2.2 Description informelle

La version historique peut facilement être décrite en se plaçant du point de vue d'une particule. Au départ de l'algorithme, un essaim est réparti au hasard dans l'espace de recherche, chaque particule ayant également une vitesse aléatoire. Ensuite, à chaque pas de temps :

- chaque particule est capable d'évaluer la qualité de sa position et de garder en mémoire sa meilleure performance, c'est-à-dire la meilleure position qu'elle a atteinte jusqu'ici (qui peut en fait être parfois la position courante) et sa qualité (la valeur en cette position de la fonction à optimiser).
- chaque particule est capable d'interroger un certain nombre de ses congénères (ses informatrices, dont elle-même) et d'obtenir de chacune d'entre elles sa propre meilleure performance (et la qualité afférente).
- à chaque pas de temps, chaque particule choisit la meilleure des meilleures performances dont elle a connaissance, modifie sa vitesse en fonction de cette information et de ses propres données et se déplace en conséquence.

Le premier point se comprend facilement, mais les deux autres nécessitent quelques précisions. Les informatrices sont définies une fois pour toutes de la manière suivante (cf. Figure 2) :

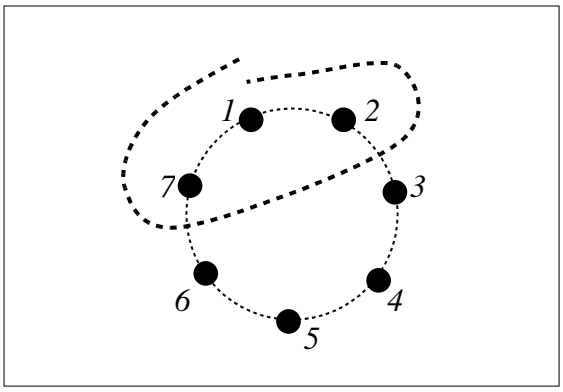


Figure 2. *Le cercle virtuel pour un essaim de sept particules. Le groupe d'information de taille trois de la particule 1 est composé des particules 1, 2 et 7.*

On suppose toutes les particules disposées (symboliquement) en cercle et, pour la particule étudiée, on inclut progressivement dans ses informatrices, d'abord elle-même, puis les plus proches à sa droite et à sa gauche, de façon à atteindre le total requis. Il y a bien sûr de nombreuses variantes, y compris celle consistant à choisir les informatrices au hasard, mais celle-ci est à la fois simple et efficace.

Une fois la meilleure informatrice détectée, la modification de la vitesse est une simple combinaison linéaire de trois tendances, à l'aide de coefficients de confiance :

- la tendance « aventureuse », consistant à continuer selon la vitesse actuelle,

- la tendance « conservatrice », ramenant plus ou moins vers la meilleure position déjà trouvée,
- la tendance « panurgienne », orientant approximativement vers la meilleure informatrice.

Les termes « plus ou moins » ou « approximativement » font référence au fait que le hasard joue un rôle, grâce à une modification aléatoire limitée des coefficients de confiance, ce qui favorise l'exploration de l'espace de recherche. La Figure 3 présente un schéma de principe résumant les explications ci-dessus. Naturellement, pour pouvoir être programmé, tout ceci est formalisé dans des équations de mouvement. Un point intéressant est que, contrairement à bien d'autres heuristiques qui restent purement expérimentales, il existe une analyse mathématique précisant les conditions de convergence et le choix des paramètres .

2.3 Principales caractéristiques

Ce modèle présente quelques propriétés intéressantes, qui en font un bon outil pour de nombreux problèmes d'optimisation, particulièrement les problèmes fortement non linéaires, continus ou mixtes (certaines variables étant réelles et d'autres entières) :

- il est facile à programmer, quelques lignes de code suffisent dans n'importe quel langage évolué,
- il est robuste (de mauvais choix de paramètres dégradent les performances, mais n'empêchent pas d'obtenir une solution).

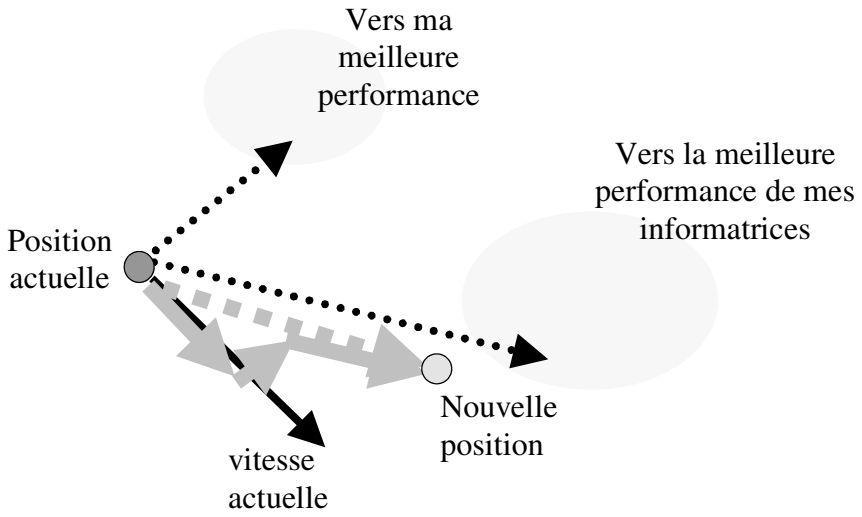


Figure 3. *Schéma de principe du déplacement d'une particule. Pour réaliser son prochain mouvement, chaque particule combine trois tendances : suivre sa vitesse propre, revenir vers sa meilleure performance, aller vers la meilleure performance de ses informatrices.*

Signalons, de plus, qu'il existe des versions adaptatives qui évitent même à l'utilisateur la peine de définir les paramètres (taille de l'essaim, taille des groupes d'informatrices, coefficients de confiance) . L'une de ces méthodes (Tribes) est décrite en détail dans un des documents téléchargeables à partir du site du séminaire OEP'03. Son code source est disponible via le site *Particle Swarm Central* [PSC]. Par ailleurs, on notera que cette heuristique se démarque des autres méthodes évolutionnaires (typiquement, les algorithmes

génétiques) sur deux points essentiels : elle met l'accent sur la coopération plutôt que sur la compétition et il n'y a pas de sélection (au moins dans les versions de base), l'idée étant qu'une particule même actuellement médiocre mérite d'être conservée, car c'est peut-être justement elle qui permettra le succès futur, précisément du fait qu'elle sort des sentiers battus.

2.4 Formalisation et programmation

La version de base de l'OEP peut facilement être formalisée et programmée. L'espace de recherche est de dimension *D*. La position courante d'une particule dans cet espace à l'instant *t* est donnée par un vecteur $x(t)$, à *D* composantes, donc. Sa vitesse courante est $v(t)$. La meilleure position trouvée jusqu'ici par cette particule est donnée par un vecteur $p(t)$. Enfin, la meilleure de celles trouvées par les informatrices de la particule est indiquée par un vecteur $g(t)$. En général, nous écrirons simplement *x*, *v*, *p* et *g*. La d$^{\text{ième}}$ composante de l'un quelconque de ces vecteurs est indiquée par l'indice *d*, par exemple x_d. Avec ces notations, les équations de mouvement d'une particule sont, pour chaque dimension *d* :

$$v_d \leftarrow c_1 v_d + alea(0, c_{max})(p_d - x_d) + alea(0, c_{max})(g_d - x_d)$$
$$x_d \leftarrow x_d + v_d$$

Pour une bonne convergence, les valeurs de c_1 et $c_{\max}$ ne doivent pas être choisies indépendamment . En pratique le premier doit être un peu inférieur à 1 et le second peut être calculé par la formule $c_{\max} = (2/0,97725)\, c_1$. Plus c_1 est proche de 1 meilleure est l'exploration de l'espace de recherche, au détriment, néanmoins, de la vitesse de convergence. Le petit programme en langage C donné en Annexe utilise un tel couple de valeurs. Il inclut également une notion de confinement dont il convient de dire quelques mots.

Lors de l'évolution de l'essaim, il peut arriver qu'une particule sorte de l'espace de recherche initialement défini. C'est sans importance si la valeur de sa position est encore calculable sans « planter » le programme informatique, mais il suffit, par exemple, d'avoir à évaluer la racine d'un nombre négatif ou une division par zéro pour que cela pose problème. Plus généralement, on souhaite souvent rester dans un espace de recherche fini donné. Par conséquent, on ajoute un mécanisme pour éviter qu'une particule ne sorte de cet espace. Le plus fréquent est le *confinement d'intervalle*. Supposons, par simplicité, que l'espace de recherche soit $\left[x_{\min}, x_{\max}\right]^D$. Alors ce mécanisme stipule que si une coordonnée x_d, calculée selon les équations de mouvement, sort de l'intervalle $\left[x_{\min}, x_{\max}\right]$, on lui attribue en fait la valeur du point frontière le plus proche. En pratique, cela revient donc à remplacer la deuxième ligne des équations de mouvement par

$$x_d \leftarrow MIN\left(MAX\left(x_d + v_d, x_{\min}\right), x_{\max}\right)$$

De plus, on complète souvent le mécanisme de confinement par une modification de la vitesse, soit en remplaçant la composante qui pose problème par son opposée, souvent pondérée par un coefficient inférieur à 1, soit, tout simplement, en l'annulant.

Plus généralement, le *principe* même du confinement consiste à dire ceci : « si une particule tend à sortir de l'espace de recherche, alors la ramener au point le plus proche qui soit dans cet espace et modifier sa vitesse en conséquence ». Il permet de définir les confinements

nécessaires à des problèmes à granularité non nulle (positions à valeurs entières, par exemple) ou à des problèmes (combinatoires, en général) dont les solutions doivent avoir toutes les coordonnées différentes.

2.5 Un petit exemple : recherche du champ minimum entre deux stations

Voici un exemple très simplifié s'inspirant d'un problème réel. On considère deux stations A et B, émettrices d'un champ électromagnétique sinusoïdal décroissant avec la distance. On cherche le point entre ces deux stations où le champ total reçu à un instant donné est minimum. La figure 4 donne une idée des champs en tout point entre les deux stations, distantes de 2,7 km. En posant x comme étant la distance à la station A, on a les équations suivantes :

$$\text{champ\_A} = e^{-x}\left(1+\sin\left(10\,x\right)\right)$$

$$\text{champ\_B} = e^{-|2,7-x|}\left(1+\sin\left(10\,|2,7-x|\right)\right)$$

On cherche le point x où la somme champ_A+champ_B est minimale. On remarquera que la longueur d'onde commune n'étant pas en rapport simple avec la distance entre les stations, la détermination analytique n'est pas évidente.

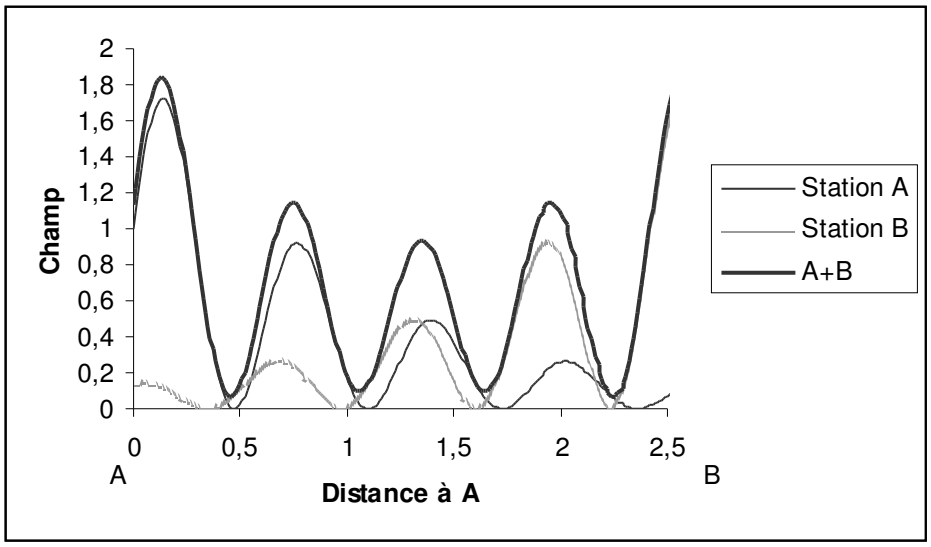


Figure 4. *Entre les deux stations A et B, à un instant donné, les valeurs des champs reçus se répartissent selon des sinusoïdes amorties. La distance entre les stations (ici 2,7 km) n'étant pas en rapport simple avec la longueur d'onde, le point où le champ total est minimum n'est pas facilement calculable.*

De plus, les méthodes classiques (gradients, par exemple) échouent du fait de l'existence de multiples minimums locaux. Par contre, il est très facile d'ajouter la fonction à minimiser au programme d'OEP donné en Annexe. Le problème est de dimension 1, et l'espace de recherche est [0 2,7]. En moins de 200 évaluations, on trouve $x = 0{,}4555478$ et, donc, un champ total de 0,06735316. C'est déjà une très bonne approximation du minimum global. Une solution un peu meilleure peut être obtenue avec plus d'itérations, mais le gain est très minime. Par exemple, après 20000 évaluations, on a $x = 0{,}4553560$ et un champ total de 0,06735263.

A titre de comparaison, avec le solveur d'un tableur courant du marché, on ne trouve pas du tout la solution si l'on indique le même espace de recherche. C'est seulement si l'on donne un point de départ déjà proche (par exemple 0,4) que le minimum est trouvé. Ceci suggère que, intégrée à d'autres logiciels, l'OEP pourrait être un solveur particulièrement efficace.

2.6 Pour en savoir plus sur l'optimisation par essaim particulaire

A ce jour, environ 250 articles sur le sujet ont été publiés, mais très peu en français . Un excellent point d'entrée est le site *Particle Swarm Central* , avec une bibliographie très complète, des liens vers des documents et des programmes à télécharger, ainsi qu'une liste de chercheurs travaillant dans ce domaine. Le livre *Swarm Intelligence* , écrit par les deux concepteurs de la méthode, y consacre quelques chapitres et donne des aperçus plus généraux sur les questions d'intelligence collective.

2.7 Séminaire OEP'2003 (Paris, 2 octobre 2003)

Le groupe de travail META (*Métaheuristiques : théorie et applications*), groupe de travail commun au GdR ALP et au GdR MACS du CNRS, a organisé, avec le soutien de la Section Automatique du Club EEA, le premier séminaire francophone OEP'2003 sur le thème de l' « Optimisation par Essaim Particulaire ».

Nous indiquons ci-dessous le programme de la journée, avec un résumé succinct de chaque exposé :

Tutoriel « Optimisation par Essaim Particulaire », par Maurice Clerc : Après un rapide historique, M. Clerc a détaillé les principes de base de l'OEP, en insistant sur les notions de coopération et de voisinages. La version « classique », sur domaine de recherche continu et qui peut être codée en quelques lignes seulement, a été analysée et discutée, ce qui a mené à l'étude mathématique des critères de convergence. L'auteur a dégagé ensuite les pré-requis minimums à l'application de la méthode, ce qui permet de la généraliser aux domaines discrets ou mixtes, au multiobjectif et aux problèmes combinatoires. Il a montré que ces derniers se situent aux frontières de la « niche écologique » de l'OEP, en ce sens qu'elle ne peut les traiter efficacement à elle seule. Puis M. Clerc a commenté les principales diversifications, en insistant sur les méthodes adaptatives permettant de s'affranchir, en partie ou totalement, du paramétrage par l'utilisateur. Diverses applications réelles ont été évoquées en commentant des articles publiés et M. Clerc a terminé par l'indication des principales sources d'information.

L'essaim de particules vu comme un système dynamique : convergence et choix des paramètres, par Ioan Cristian Trelea : Les utilisateurs de l'algorithme d'optimisation par essaim de particules (OEP) savent bien que le choix des paramètres a une influence directe sur les performances. Un éclairage intéressant est apporté en assimilant l'essaim à un système dynamique, les positions et les vitesses des particules constituant l'état du système. Pour l'essaim déterministe, la théorie des systèmes dynamiques permet de construire facilement, dans l'espace des paramètres, des cartes bidimensionnelles, où apparaissent le domaine de

convergence, et certains domaines de comportement dynamique des particules (mouvement asymptotique, oscillations harmoniques, zigzags). Pour l'essaim stochastique, ces cartes restent valables qualitativement, et donnent des indications sur le choix des paramètres, en fonction du comportement désiré de l'essaim.

Comparaison de l'optimisation par essaim particulaire avec d'autres heuristiques évolutionnaires utilisant des opérateurs implicites, par Louis Gacôgne : L'auteur a présenté une comparaison sur un certain nombre de fonctions de test, entre différentes heuristiques évolutionnaires, qui ont en commun de ne pas prendre aléatoirement des opérateurs génétiques au hasard, mais d'appliquer des règles de transition fixes entre générations :

- « particle swarm optimization » (PSO),
- « space partition algorithm » (SPA),
- « macroevolutionary algorithm » (MGA),
- « évolution différentielle » (DE),
- « self organizing migration algorithm » (SOMA).

Puis il a comparé ces heuristiques avec quelques algorithmes évolutionnaires classiques, notamment SSGA.

Diverses techniques d'optimisation inspirées de la théorie de l'auto-organisation dans les systèmes biologiques, par Johann Dréo et Patrick Siarry : Une contribution importante de la biologie vient de la *théorie de l'auto-organisation*, qui permet d'expliquer les propriétés de plusieurs métaheuristiques issues des métaphores biologiques. Cette théorie décrit les conditions d'émergence de phénomènes complexes à partir de systèmes distribués, dont les agents font l'objet d'interactions simples, mais nombreuses. L'intelligence en essaim est ainsi née sur deux fronts : via une approche « systèmes auto-organisés » (ayant donné lieu aux algorithmes de colonies de fourmis) et via une approche « systèmes socio-cognitifs » (ayant donné lieu à l'optimisation par essaims particulaires). En se plaçant dans le cadre de la théorie de l'auto-organisation, les auteurs ont conçu un algorithme inspiré des colonies de fourmis, qui insiste sur l'importance des canaux de communication.

Atelier « Optimisation par Essaim Particulaire », par Maurice Clerc : Cet atelier a permis de montrer, sur des exemples, le « domaine de compétence » de l'OEP, et de discuter sur des problèmes pour lesquels la méthode ne donne pas de bons résultats.

Les transparents et les textes d'accompagnement des différents exposés peuvent être téléchargés à partir du site du séminaire : http://www.particleswarm.net/oep_2003/.

Références

[Berthiau et al. 01] Berthiau G., Siarry P., « Etat de l'art des méthodes d'optimisation *globale* », RAIRO-Operations Research, Vol. 35, N°3, p. 329-365, 2001.

[Bonabeau et al. 99] Bonabeau E., Dorigo M., Theraulaz G., « Swarm Intelligence : from natural to artificial systems ». Oxford University Press, 1999.

[Clerc et al. 02] Clerc M., Kennedy J., « The Particle Swarm. Explosion, stability, and convergence in a multidimensional complex space », IEEE Transactions on Evolutionary Computation, Vol. 6, p. 58-73, 2002.

Sites Internet

Méthode des essaims particulaires
http://www.particleswarm.net/
Séminaire OEP'2003
http://www.particleswarm.net/oep_2003/
Groupe de travail sur les métaheuristiques
http://www.lifl.fr/~talbi/META
Livre récent sur les métaheuristiques
http://www.eyrolles.com/php.informatique/Ouvrages/ouvragem.php3?ouv_ean13=9782212113686

Annexe : **code source en C d'une version simple d'OEP**

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>

#define D_max 100 // Dimension maximale de l'espace de recherche

struct position     {int taille;double x[D_max]; double f;}; // Position d'une particule et sa performance
struct vecteur      {int taille;double v[D_max];}; // Vitesse d'une particule

// Sous-programmes
double   alea(double a,double b); // Nombre pseudo-aléatoire entre a et b
double   ma_fonction (struct position x, int fonction); // Fonctions à minimiser

// Variables globales
int         nb_eval; // Nombre total d'évaluations
//============================================== PROGRAMME PRINCIPAL
void main()
{
double              c1,cmax;          // Coefficients de confiance
int                 D;                // Dimension de l'espace de recherche
int                 d;                // Numéro de la dimension courante
double              eps;              // Précision souhaitée
int                 eval_max;         // Nombre max d'évaluations de la fonction
double              eval_moyen;       // Nombre moyen d'évaluations
double              fmin;             // Objectif à atteindre
int                 fonction;         // Numéro de la fonction à minimiser (cf. sous_programme ma_fonction)
int                 g;                // Numéro de la meilleure informatrice
int                 k;
struct position     P[D_max];         // Positions
struct position     P_m[D_max];       // Meilleures positions trouvées
struct position     meilleure;        // Toute meilleure position trouvée
int                 K;                // Taille des groupes d'informatrices
double              min_f;
int                 N;                // Taille de l'essaim
int                 n;                // Numéro de la particule courante
int                 n_exec,n_exec_max; // Pour exécutions multiples
int                 rang;
int                 signe;
struct vecteur      V[D_max];         // Vitesses
double              xmin,xmax;        // Intervalle pour l'espace de recherche

// Paramètres de réglage
c1=0.738; // Confiance en la tendance actuelle
cmax=1.51; // Confiance maximale en les informatrices
N=20; // Taille de l'essaim
K=3; // Nombre de liens d'information

// Problème à traiter (à changer selon le problème, évidemment)
fonction=4; // Cf. sous-programme ma_function()
xmin=-10; xmax=10; D=2; // Espace de recherche
eps=0.00001; // Précision souhaitée
fmin=0; // Valeur minimale à atteindre, à la précision près
eval_max=40000; // Nombre maximum d'évaluations
n_exec_max=20; // Nombre d'exécutions
//-------------------------------------------------------------------------------
n_exec=0;
```

```
eval_moyen=0;

init: // Initialisations des positions et vitesses

for (n=0;n<N;n++)
{
	P[n].taille=D;
	for (d=0;d<D;d++) P[n].x[d]=alea(xmin,xmax);

	V[n].taille=D;
	for (d=0;d<D;d++) V[n].v[d]=alea((xmin-xmax)/2,(xmax-xmin)/2);
}

// Evaluations initiales
nb_eval=0;

for (n=0;n<N;n++)
{
	P[n].f=ma_fonction(P[n],fonction); // Evaluation de la position
	P_m[n]=P[n]; // Meilleure position = position initiale
}

// Mémorisation du meilleur résultat atteint jusqu'ici
meilleure=P_m[0];
for (n=0;n<N;n++) if (P_m[n].f<meilleure.f) meilleure=P_m[n];

boucle: // Itérations

for (n=0;n<N;n++) // Pour chaque particule
{
	// Meilleure informatrice dans le i-groupe circulaire de taille K
	g=n; min_f=P_m[n].f;
	signe=1; k=1;
autre_informatrice:
		rang=n+signe*k;
		if (rang>N-1) rang=rang-N+1;
		if (rang<0) rang=rang+N-1;
		if (P_m[rang].f<min_f) {g=rang;min_f=P_m[rang].f;}
		if (k<K) {signe=-signe;k=k+1; goto autre_informatrice;}

	// Calcul de la nouvelle vitesse
	for (d=0;d<D;d++) V[n].v[d]=c1*V[n].v[d]+alea(0,cmax)*(P_m[n].x[d]-
P[n].x[d])+alea(0,cmax)*(P_m[g].x[d]-P[n].x[d]);

	// Déplacement
	for (d=0;d<D;d++) P[n].x[d]=P[n].x[d]+V[n].v[d];

	// Confinement d'intervalle
	for (d=0;d<D;d++)
	{
		if (P[n].x[d]<xmin) {P[n].x[d]=xmin;V[n].v[d]=0;} // ou bien V[n].v[d]=-0.5* V[n].v[d]
		if (P[n].x[d]>xmax) {P[n].x[d]=xmax;V[n].v[d]=0;} // ou bien V[n].v[d]=-0.5* V[n].v[d]
	}

	// Evaluation de la nouvelle position
	P[n].f=ma_fonction(P[n],fonction);

	// Mise à jour de la meilleure position
```

```
            if (P[n].f<P_m[n].f) P_m[n]=P[n];

            // Mémorisation du meilleur résultat atteint jusqu'ici
            if (P_m[n].f<meilleure.f) meilleure=P_m[n];

}


// Test de fin
if (fabs(meilleure.f-fmin)>eps && nb_eval<eval_max) goto boucle;

// Affichage du meilleur résultat trouvé
printf("\n Eval= %i. Meilleure position (valeur %f).  : \n",nb_eval,meilleure.f);
for (d=0;d<D;d++) printf(" %f",meilleure.x[d]);

//  Boucle d'itération
n_exec=n_exec+1;
eval_moyen=eval_moyen+nb_eval;
if (n_exec<=n_exec_max) goto init;

// Résultat moyen
eval_moyen=eval_moyen/n_exec;
printf("\n Eval moyen = %f",eval_moyen);
}
//================================================= ALEA
double   alea(double a,double b)
{ // Délivre un nombre pseudo-aléatoire entre a et b selon une distribution simulant l'uniforme
double   r;

// Normalement, RAND_MAX = 32767 = 2^15-1

            r=(double)rand()/RAND_MAX; // r dans [0,1]
            r=a+r*(b-a);
            return r;
}
//================================================= MA_FONCTION
double ma_fonction(struct position x, int fonction)
{// Evalue la valeur de la fonction à minimiser à la position x
 // AJOUTEZ VOTRE PROPRE FONCTION
int    D,d;
double f,p,xd;

nb_eval=nb_eval+1;
D=x.taille;

switch (fonction)
{
case 1: // Sphère
            f=0;
            for(d=0;d<D;d++) f=f+x.x[d]*x.x[d];
break;

case 2: // Pour test du confinement. Utiliser un xmin >=0
            f=0;
            for(d=0;d<D;d++) f=f+sqrt(x.x[d]);
break;

case 3: // Alpine. Min 0 en (0,0 ...0)
```

```
		f=0;
		for( d=0;d<D;d++)
		{
			f=f+sqrt(fabs(x.x[d]*sin(x.x[d])));
		}
break;

case 4: // Rosenbrock, fonction Banane. Min 0 en (1, ...1)
		f=0;
		for (d=0;d<D-1;d++)
		{
			xd=1-x.x[d];
			f=f+xd*xd;
			xd= x.x[d]*x.x[d]-x.x[d+1];
			f=f+100*xd*xd;
		}
		break;
}
return f;
}
```